The Chombo Library for Adaptive Mesh Refinement Applications

Phillip Colella
Lawrence Berkeley National Laboratory

Local Refinement for Partial Differential Equations

Variety of problems that exhibit multiscale behavior, in the form of localized large gradients separated by large regions where the solution is smooth.

- Shocks and interfaces.
- Self-gravitating flows in astrophysics.
- Complex engineering geometries.
- Combustion.
- Magnetohydrodynamics: space weather, magnetic fusion.

In adaptive methods, one adjusts the computational effort locally to maintain a uniform level of accuracy throughout the problem domain.

Adaptive Mesh Refinement (AMR)

Modified equation analysis: finite difference solutions to partial differential equations behave like solutions to the original equations with a modified right-hand side.

For linear steady-state problems $LU = f$:

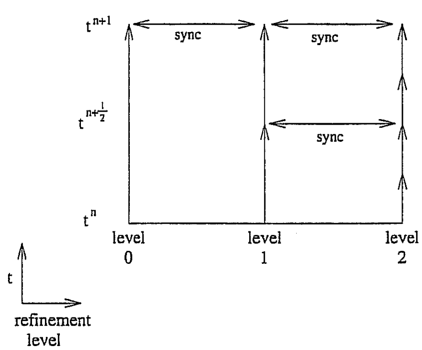
$$\epsilon = U^h - U \quad , \quad \epsilon \approx L^{-1}\tau$$

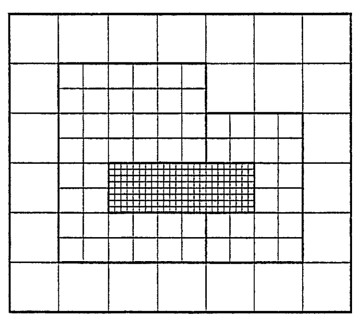
For nonlinear, time=dependent problems

$$\frac{\partial U}{\partial t} + L(U) = 0 \Rightarrow \frac{\partial U^h}{\partial t} + L(U^h) = \tau$$

In both cases, The truncation error $\tau = \tau(U) = (\Delta x)^p M(U)$, where M is a $(p+q)$-order differential operator.

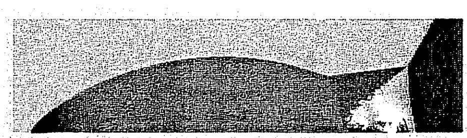
Block-Structured Local Refinement (Berger and Oliger, 1984)

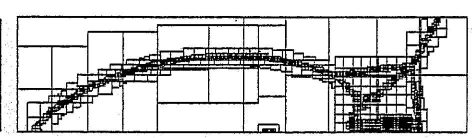




Refined regions are organized into rectangular patches

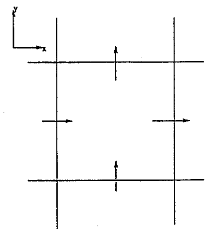
Refinement performed in time as well as in space.





AMR for Hyperbolic Conservation Laws (Berger and Colella, 1989)
We assume that the underlying uniform-grid method is an explicit conservative difference method.

$$U^{new} := U^{old} - \Delta t(D\vec{F}) \ , \ \vec{F} = \vec{F}(U^{old})$$

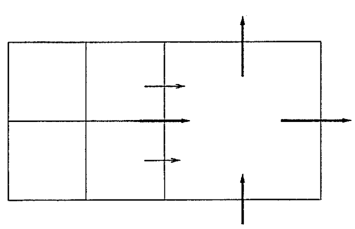


On a two-level AMR grid, we have U^c, U^f, and the update is performed in the following steps.

- Update solution on entire coarse grid: $U^c := U^c - \Delta t^c D^c \vec{F}^c$.
- Update solution on entire fine grid: $U^f := U^f - \Delta t^f D^f \vec{F}^f$ (n_{refine} times).
- Synchronize coarse and fine grid solutions.

Synchronization of Multilevel Solution

- Average coarse-grid solution onto fine grid.
- Correct coarse cells adjacent to fine grid to maintain conservation.



$$U^c := U^c + \Delta t^c (F^{c,s}_{i^c - \frac{1}{2}e} - \frac{1}{Z} \sum_{i^f} F^{f,s}_{i^f - \frac{1}{2}e})$$

Typically, need a generalization of GKS theory for free boundary problem to guarantee stability (Berger, 1985). Stability not a problem for upwind methods.

Software Approach

Requirement: to support a wide variety of applications that use block-structured AMR using a common software framework.

- Mixed-language model: C++ for higher-level data structures, Fortran for regular single-grid calculations.

- Reuseable components. Component design based on mapping of mathematical abstractions to classes.

- Build on public-domain standards: MPI, HDF5, VTK.

- Interoperability with other SciDAC ISIC tools: grid generation (TSTT), solvers (TOPS), performance analysis tools (PERC).

Previous work: BoxLib (LBNL/CCSE), KeLP (Baden, et. al., UCSD), FIDIL (Hilfinger and Colella).

Layered Design

- **Layer 1.** Data and operations on unions of boxes – set calculus, rectangular array library (with interface to Fortran), data on unions of rectangles, with SPMD parallelism implemented by distributing boxes over processors.

- **Layer 2.** Tools for managing interactions between different levels of refinement in an AMR calculation – interpolation, averaging operators, coarse-fine boundary conditions.

- **Layer 3.** Solver libraries – AMR-multigrid solvers, Berger-Oliger time-stepping.

- **Layer 4.** Complete parallel applications.

- **Utility layer.** Support, interoperability libraries – API for HDF5 I/O, visualization package implemented on top of VTK, C API's.

Examples of Layer 1 Classes (BoxTools)

• IntVect $\boldsymbol{i} \in \mathbb{Z}^d$. Can translate $\boldsymbol{i}_1 \pm \boldsymbol{i}_2$, coarsen $\frac{\boldsymbol{i}}{s}$, refine $\boldsymbol{i} * s$.

• Box $B \subset \mathbb{Z}^d$ is a rectangle: $B = [\boldsymbol{i}_{low}, \boldsymbol{i}_{high}]$. B can be translated, coarsened, refined. Supports different centerings (node-centered vs. cell-centered) in each coordinate direction.

• IntVectSet $\mathcal{I} \subset \mathbb{Z}^d$ is an arbitrary subset of $\mathbb{Z}^d$. $\mathcal{I}$ can be shifted, coarsened, refined. One can take unions and intersections, with other IntVectSets and with Boxes, and iterate over an IntVectSet.

• FArrayBox A(Box B, int nComps): multidimensional arrays of Reals constructed with B specifying the range of indices in space, nComp the number of components. Real* FArrayBox::dataPointer returns pointer to the contiguous block of data that can be passed to Fortran.

Example: explicit heat equation solver on a single grid

```
// C++ code:

  Box domain(-IntVect:Unit,nx*IntVect:Unit);
  FArrayBox soln(grow(domain,1), 1);
  soln.setVal(1.0);

  for (int nstep = 0;nstep < 100; nstep++)
 {
  heatsub2d_(soln.dataPtr(0),
            &(soln.loVect()[0]), &(soln.hiVect()[0]),
            &(soln.loVect()[1]), &(soln.hiVect()[1]),
              region.loVect(), region.hiVect(),
              &dt, &dx, &nu);
 }
```

```
c Fortran code:
      subroutine heatsub2d(phi,nlphi0, nhphi0,nlphi1, nhphi1,
     &     nlreg, nhreg, dt, dx, nu)

      real*8 lphi(nlphi0:nhphi0,nlphi1:nhphi1)
      real*8  phi(nlphi0:nhphi0,nlphi1:nhphi1)
      real*8 dt,dx,nu
      integer nlreg(2),nhreg(2)

c Remaining declarations, setting of boundary conditions goes here.
...
       do j = nlreg(2), nhreg(2)
         do i = nlreg(1), nhreg(1)
            lapphi =
     &           (phi(i+1,j)+phi(i,j+1)
     &           +phi(i-1,j)+phi(i,j-1)
     &           -4.0d0*phi(i,j))/(dx*dx)

            lphi(i,j) = lapphi
         enddo
       enddo
```

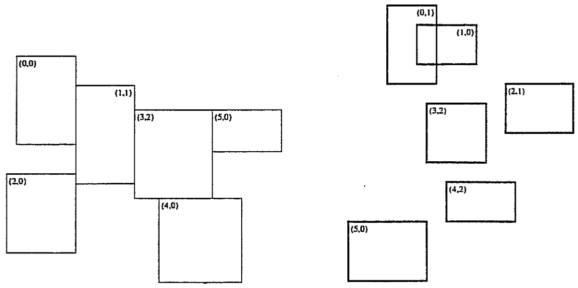
```
c Increment solution with rhs.

       do j = nlreg(2), nhreg(2)
          do i = nlreg(1), nhreg(1)
             phi(i,j) = phi(i,j) + nu*dt*lphi(i,j)
          enddo
       enddo

       return
       end
```

Distributed Data on Unions of Rectangles

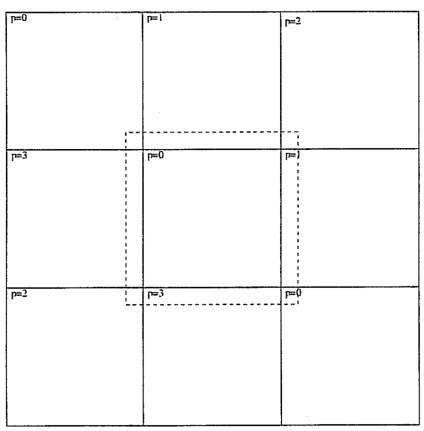
Provides a general mechanism for distributing data defined on unions of rectangles onto processors, and communications between processors.



• Metadata of which all processors have a copy: BoxLayout is a collection of Boxes and processor assignments: $\{B_k, p_k\}_{k=1}^{nGrids}$. DisjointBoxLayout:public BoxLayout is a BoxLayout for which the Boxes must be disjoint.

• template <class T> LevelData<T> and other container classes hold data distributed over multiple processors. For each k = 1 ... nGrids, an "array" of type T corresponding to the box B_k is allocated on processor p_k. Straightforward API's for copying, exchanging ghost cell data, iterating over the arrays on your processor in a SPMD manner.

Example: explicit heat equation solver, parallel case



Want to apply the same algorithm as before, except that the data for the domain is decomposed into pieces and distributed to processors.

• LevelData<T>::exchange(): obtains ghost cell data from valid regions on other patches.

• DataIterator: iterates over only the patches that are owned on the current processor.

```
// C++ code:
  Box domain;
  DisjointBoxLayout dbl;
// Break domain into blocks, and construct the DisjointBoxLayout.
  makeGrids(domain,dbl,nx);

  LevelData<FArrayBox> phi(dbl, 1, IntVect::TheUnitVector());

  for (int nstep = 0;nstep < 100;nstep++)
  {
...
// Apply one time step of explicit heat solver: fill ghost cell valu
// and apply the operator to data on each of the Boxes owned by this
// processor.

  phi.exchange();
  DataIterator dit = dbl.dataIterator();

// Iterator iterates only over those boxes that are on this processc





  for (dit.reset();dit.ok();++dit)
  {
  FArrayBox& soln = phi[dit()];
  Box& region = dbl[dit()];
  heatsub2d_(soln.dataPtr(0),
            &(soln.loVect()[0]), &(soln.hiVect()[0]),
            &(soln.loVect()[1]), &(soln.hiVect()[1]),
              region.loVect(), region.hiVect(),
              domain.loVect(), domain.hiVect(),
              &dt, &dx, &nu);
  }
  }
```

Load Balancing

For parallel performance, need to obtain approximately the same work load on each processor.

• Unequal-sized grids: knapsack algorithm provides good efficiencies provided the number of grids / processor ≥ 3 (Crutchfield, 1993). Disadvantage: does not preserve locality.

• Equal-sized grids can provide perfect load balancing if algorithm is reasonably homogeneous. Disadvantage: many small patches can lead to large amounts of redundant work.

Both methods obtain good scaling into 100's of nodes for hyperbolic problems. Alternative approach: space-filling curves using equal-sized grids, followed by agglomeration.

Software Reuse by Templating Dataholders

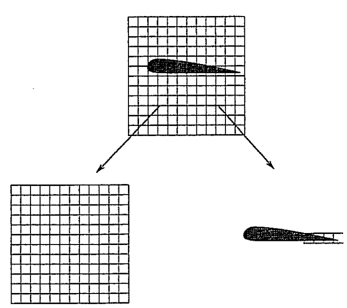
Classes can be parameterized by types, using the class template language feature in C++.

`BaseFAB<T>` is a multidimensional array whihc can be defined for for any type `T`. `FArrayBox:  public BaseFAB<Real>`

In `LevelData<T>`, `T` can be any type that "looks like" a multidimensional array. Examples include:

• Ordinary multidimensional arrays, e.g. `LevelData<FArrayBox>`.

• A composite array type for supporting embedded boundary computations:



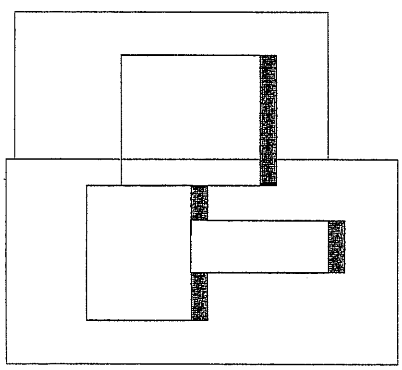
• Binsorted lists of particles, e.g. `BaseFab<List<ParticleType>>`

Layer 2: Coarse-Fine Interactions (`AMRTools`).
The operations that couple different levels of refinement are among the most difficult to implement AMR.

- Interpolating between levels (`FineInterp`).
- Interpolation of boundary conditions (`PWLFillpatch, QuadCFInterp`).
- Averaging down onto coarser grids (`CoarseAverage`).
- Managing conservation at coarse-fine boundaries (`LevelFluxRegister`).

These operations typically involve interprocessor communication and irregular computation.

Example: `class LevelFluxRegister`



$$U^c := U^c + \Delta t^c (F^{c,s}_{i^c - \frac{1}{2}e} - \frac{1}{Z}\sum_{i^f} F^{f,s}_{i^f - \frac{1}{2}e})$$

The coarse and fine fluxes are computed at different times in the program, and on different processors. We rewrite the process in the following steps:

$$\delta F = 0$$

$$\delta F := \delta F - \Delta t^c F^c$$

$$\delta F := \delta F + \Delta t^f < F^f >$$

$$U^c := U^c + D_R(\delta F)$$

A `LevelFluxRegister` object encapsulates these operations:

- `LevelFluxRegister::setToZero()`

- `LevelFluxRegister::incrementCoarse`: given a flux in a direction for one of the patches at the coarse level, increment the flux register for that direction.

- `LevelFluxRegister::incrementFine`: given a flux in a direction for one of the patches at the fine level, increment the flux register with the average of that flux onto the coarser level for that direction.

- `LevelFluxRegister::reflux`: given the data for the entire coarse level, increment the solution with the flux register data for all of the coordinate directions.

Layer 3: Reusing Control Structures Via Inheritance (`AMRTimeDependent`, `AMRElliptic`).

AMR has multilevel control structures are largely independent of the details of the operators and the data.

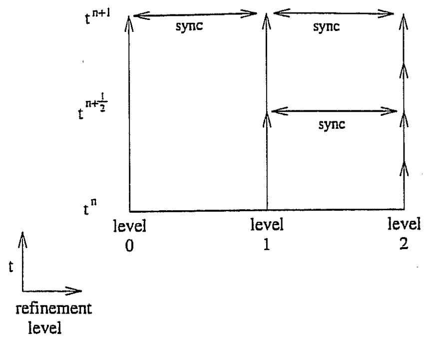
- Berger-Oliger refinement in time.

- Multigrid iteration on a union of rectangles.

- Multigrid iteration on an AMR hierarchy.

To separate the control structure from the details of the operations that are being controlled, we use C++ inheritance in the form of *interface classes*.

Example: `AMR / AMRLevel` **interface for Berger-Oliger timestepping**



We implement this control structure using a pair of classes.

`class AMR`: manages the Berger-Oliger time-stepping process.

`class AMRLevel`: collection of virtual functions called by an `AMR` object that perform the operations on the data at a level, e.g.:

- `virtual void AMRLevel::advance() = 0` advances the data at a level by one time step.
- `virtual void AMRLevel::postTimeStep() = 0` performs whatever sychronization operations required after all the finer levels have been updated.

`AMR` has as member data a collection of pointers to objects of type `AMRLevel`, one for each level of refinement:

```
Vector<AMRLevel*> m_amrlevels;
```

`AMR` calls the various member functions of `AMRLevel` as it advances the solution in time:

```
m_amrlevels[currentLevel]->advance();
```

The user implements a class derived from `AMRLevel` that contains all of the functions in `AMRLevel`:

```
class AMRLevelWaveEquation : public AMRLevel
// Defines functions in the interface, as well as data.
...
virtual void AMRLevelWaveEquation::advance()
{
// Advances the solution for one time step.
...
}
```

To use the `AMR` class for this particular application, `m_amrlevel[k]` will point to objects in the derived class, e.g.,

```
AMRLevelWaveEquation* amrLevelWavePtr = new AMRLevelWaveEquation(...);
m_amrlevel[k] = static_cast <AMRLevel*> (amrWavePtr);
```

Layer 4: AMR Applications

- A general driver for an unsplit second-order Godunov method for hyperbolic conservation laws. User provides physics-dependent components (characteristic analysis, Riemann solver).

- Level solvers, AMR multigrid solvers for constant-coefficient Poisson, Helmholtz equations. Variable-coefficient elliptic solvers under development.

- Incompressible Navier-Stokes solver using projection method. Includes projection operators for single level, AMR hierarchy. Advection-diffusion solvers.

- Wave equation solver.

- Landau-Ginzburg solver.

- Volume-of-fluid algorithm fluid-solid interactions.

AMR Utility Layer

- API for HDF5 I/O.

- Interoperability tools. We are developing a framework-neutral representation for pointers to AMR data, using opaque handles. This will allow us to wrap Chombo classes with a C interface and call them from other AMR applications.

- Chombo Fortran - a macro package for writing dimension-independent Fortran and managing the Fortran / C interface.

- `Parmparse` class from BoxLib for handling input files.

- Visualization and analysis tools (ChomboVis).

Current Status and Future Plans

Chombo version 1.2 is available at the ANAG web site:
`http://seesar.lbl.gov/anag/software.html`

Chombo requires gmake, perl, a Fortran 77 compiler, and a reasonably standards-compliant C++ compiler (gcc 2.95.3 or later, except 2.96).

Distribution includes source code for the libraries, design documentation, an html reference manual, and a number of examples: heat equation on a single grid, AMR gas dynamics, AMR Poisson. Other examples soon to be released: AMR for heat equation, wave equation, incompressible Navier-Stokes.

New Capabilities. We have a number of new capabilities under development, that will be released sometime in the next year.

- Embedded boundary treatment of geometry.
- AMR and particles (PIC, PPPM).

Acknowledgements

DOE Applied Mathematical Sciences Program
DOE HPCC Program
DOE SciDAC Program
NASA Earth and Space Sciences Computational Technologies Program

ChomboVis Interactive Visualization and Analysis Tools

- Block-structured representation of the data leads to efficiency.
- Visualization tools based on VTK, a public-domain visualization library.
- Implementation in Python provides command-line interface to visualization and analysis tools, batch processing capability.
- Interface to HDF5 I/O provides access to broad range of AMR users.